
Random Walk Foton di Zona Radiasi Matahari

Febrie Ahmad Azizi || 20912008

Ridlo W. Wibowo || 20912009

December 15, 2012

Dokumentasi.
Program untuk menyimulasikan pergerakan random walk foton di Zona Radiasi Matahari kami buat dengan sederhana. Algoritmanya dapat dijelaskan sebagai berikut:

1. Inisiasi, $r = 0$ atau $(0, 0, 0)$, dan $t = 0$
2. Acak *uniform* arah dalam koordinat Bola.
3. Acak tempat terjadi *collision* (r_{col}) dengan mengunakan aturan yang akan dijelaskan dibawah (*tracking*).
4. Update waktu foton berjalan (t).
5. Tentukan posisi baru foton dengan transformasi koordinat kartesian menggunakan r_{col} dan arah geraknya.
6. Hitung dan update r baru.
7. Bandingkan r dengan R_{rad}, jika $r \geq R_{rad}$ STOP.
8. Ulangi ke langkah no 2.
9. Hitung perbandingan waktu cahaya berjalan secara *randomwalk* dengan foton berjalan lurus sejauh R_{rad}, kita sebut sebagai $t_{mode,scale} = \frac{t}{R_{rad}/c}$, $c =$ kecepatan cahaya.

Source

Untuk membuat random arah yang *uniform* saat awal simulasi ($r = 0$) dan arah baru setelah terjadi tumbukan, maka dibuat random dalam koordinat bola. Posisi atau vektor dalam ruang 3D dapat dinyatakan dengan koordinat Bola (r, ϕ, θ), dengan transformasi ke koordinat kartesian dapat menggunakan:

$$x = r \sin\theta \cos\phi$$

$$y = r \sin\theta \sin\phi$$

$$z = r \cos\theta$$

Luas sudut permukaan Bola adalah $\Omega = 4\pi$ steradian.

$$F(\Omega) = \int_0^{\Omega} \frac{d\Omega}{4\pi}$$

$$F(\theta, \phi) = \frac{1}{4\pi} \int_0^{\theta} \int_0^{\phi} \sin\theta d\theta d\phi$$

$$F(\theta, \phi) = \frac{1}{4\pi} \int_0^{\theta} \sin\theta d\theta \int_0^{\phi} d\phi$$

$$F(\theta, \phi) = \frac{1}{4\pi}(1 - \cos\theta)\phi$$

sehingga diperoleh,

$$F(\theta) = \frac{1}{2}(1 - \cos\theta) = r_1$$

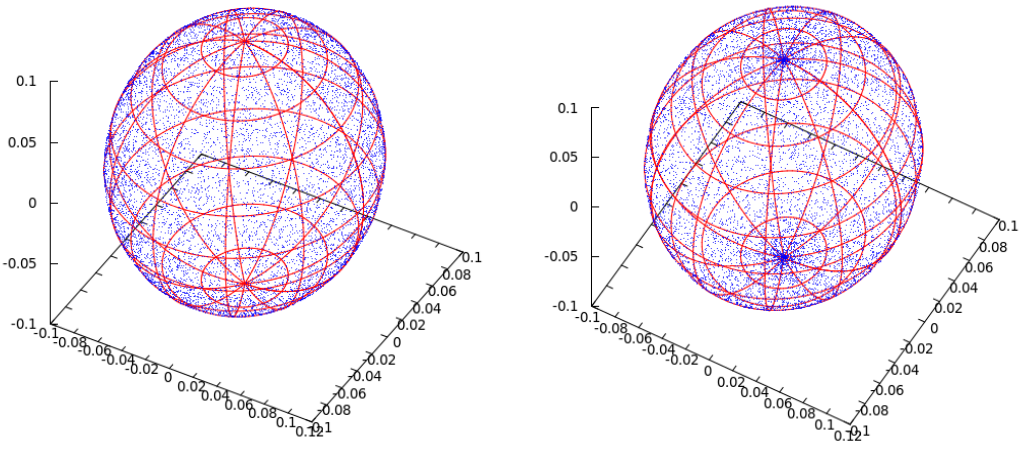
$$F(\phi) = \frac{\phi}{2\pi} = r_2$$

atau,

$$\theta = \arccos(1 - 2r_1)$$

$$\phi = 2\pi r_2$$

Perbandingan random dengan aturan di atas dan dengan mengambil random langsung untuk sudutnya $(0 < r_1 < \pi, 0 < r_2 < 2\pi)$ yang akan mengakibatkan lebih banyak titik/arah pada kutubnya ditunjukkan pada gambar di bawah ini,



(kiri) distribusi *uniform* yang benar, (kanan) distribusi tidak *uniform*.

Tracking

Bagian ini berguna untuk menentukan seberapa jauh foton bergerak sebelum menumbuk elektron, sehingga faktor utama yang mempengaruhi adalah *mean free path* dari foton saat berada di zona radiasi Matahari. Telah dijelaskan di laporan, pendekatan dilakukan untuk menyimulasikan kasus ini, karena yang kita ingin tentukan hanya waktu yang dibutuhkan foton untuk keluar dari zona radiasi. Penyederhanaan itu dilakukan dengan menggunakan teori klasik yaitu *Thomson scattering* untuk menentukan *mean free path* foton. Telah diketahui pula bahwa *cross-section* foton terhadap elektron jauh lebih besar dibandingkan dengan proton.

Mean free path (f) foton ditentukan dengan:

$$f = \frac{m_e + m_p}{\sqrt{2}\sigma_T \rho}$$

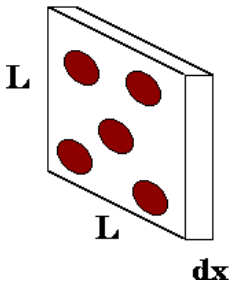
dengan σ_T merupakan *Thomson cross-section* sebesar $6.65245854533 \times 10^{-29}$ m^2, dan ρ merupakan kerapatan zona radiasi (ρ_r untuk kasus model kerapatan linear).

Kemudian kita harus menentukan sejauh apa foton dapat bergerak lurus setelah tumbukan (r_{col}), tentunya kita dapat buat random jarak yang apabila kita rata-ratakan untuk banyak percobaan harus mendekati kembali nilai f yang sudah ditentukan di atas.

Penurunan mean free path

Dapat kita bayangkan apabila terdapat sinar atau partikel yang bergerak melintasi

kumpulan partikel lain, maka penampang melintang permukaan (*slab*) yang ia lewati dapat digambarkan seperti dibawah ini.



Luas permukaan *slab* adalah L^2 dan volumenya L^2dx, sedangkan luas penampang partikel total adalah $\sigma n L^2 dx$ dengan n adalah jumlah partikel per satuan volum. Probabilitas tumbukan dengan sistem partikel untuk selang dx menjadi,

$$P(berhentisetelah dx) = \frac{areapartikel}{areaslab} = \frac{\sigma n L^2 dx}{L^2} = \sigma n dx$$

Perubahan intensitas sinar atau jumlah partikel yang menerobos adalah

$$dI = -I\sigma n dx$$

atau,

$$\frac{dI}{dx} = -I\sigma n \equiv^{def} -\frac{I}{f}$$

didefinisikan bahwa f adalah *mena free path* yaitu jarak rata-rata perjalanan sebelum partikel menumbuk partikel lain.
Dari persamaan di atas diperoleh solusi $I = I_0 e^{-x/f}$ sehingga perubahan probabilitas tumbukan untuk selang dx adalah

$$dP(x) = \frac{I(x) - I(x+dx)}{I_0} = \frac{1}{f} e^{-x/f} dx$$

dan nilai ekspektasi (atau rata-rata) dari x menjadi

$$< x > =^{def} \int_0^\infty x dP(x) = \int_0^\infty \frac{x}{f} e^{-x/f} dx = f$$

Oleh karena itu untuk mendapatkan random yang sesuai dapat diperoleh dari,

$$F(x) = \int_0^x \frac{1}{f} e^{-x/f} dx = 1 - e^{-x/f}$$

dengan demikian jarak tumbukan (r_{col}) dapat ditentukan,

$$r_{col} = x = -f \ln(1 - r_3)$$

atau, karena r_3 merupakan random *uniform* antara 0 dan 1, maka dapat pula kita ganti menjadi

$$r_{col} = -f \ln r_3$$

Program

Program dibuat dan dijalankan untuk *single-processor* dan *multi-processor* (menggunakan *OpenMPI*). Kami membagi pekerjaan yang sama untuk dijalankan di masing-masing *processor* dengan hanya membedakan *seed* bilangan randomnya (pseudo. Untuk parameter yang sama kami perlu menjalankan 1000 kali dengan *seed* random yang berbeda, sehingga kami cukup membagi rata sesuai jumlah *processor* (dalam hal ini jumlah *thread* yang tersedia). Program terlampir di bawah, sedangkan hasil simulasi dan tabel waktu komputasi diberikan di laporan.
Screenshot htop (ubuntu) untuk melihat kinerja *processor* untuk program yang sama.

htop, (atas) tanpa MPI, dan (bawah) dengan MPI untuk 4 *core* – 8 *threads*.

Lampiran Program
Model kerapatan konstan – tanpa MPI

```
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <fstream>
#include <stdlib.h>
#include <time.h>
#include <ctime>
```

```
8  #include <string>
9  #define _USE_MATH_DEFINES
10 #define c 299792458.
11 using namespace std;
12
13 double unirand(){return (double) rand()/(double) RAND_MAX;}
14
15 double randomwalk(double Rrad){
16     double r, theta, phi, x, y, z, dx, dy, dz, f, rho, dt, t, rcol, randcol
       ;
17     double me = 9.11e-31; // kg
18     double mp = 1.67e-27; // kg
19     double sigmaT = 6.65245854533e-29; // m^2
20     unsigned int n=0;
21     x = 0.0; y = 0.0; z = 0.0, t=0.0, dt=0.0;
22     rho = 15000.; // Constant Density
23     f = (me+mp)/(sqrt(2.)*sigmaT*rho);
24     do{
25             phi = 2.*M_PI*unirand(); theta = acos(1.-2.*unirand());
26             do{
27                 randcol = unirand();
28             } while (randcol == 0.0 || randcol == 1.0); // menghindari -inf
29             rcol = -f*log(randcol);
30             dt = rcol/c;
31             t = t+dt;
32             dx = rcol*sin(theta)*cos(phi);
33             dy = rcol*sin(theta)*sin(phi);
34             dz = rcol*cos(theta);
35             x=x+dx;
36             y=y+dy;
37             z=z+dz;
38             n = n+1;
39             r = sqrt(x*x + y*y + z*z);
40     } while (r <= Rrad);
41
42     double tm = t/(Rrad/c);
43     return tm;
44 }
45
46 int main(int argc, char** argv){
47     time_t tstart=time(0), tend, tim, start, finish;
48     clock_t begin, end;
49     int N = 1000;
50     double R[7]={0.05,0.1,0.2,0.5,1.0,2.0,5.0};
51
52     for (int j=0;j<7;j++){
53         // inisiasi timer
54         if (R[j] < 1.0){ begin = clock();}
55         else { start = time(0);}
56
57         // progran utama
58         char filename[64];
59         double Rrad=R[j];
60         srand(time(NULL));
```

```
61          sprintf(filename, "out-photonConst%d.txt", j);
62          ofstream out(filename);
63          for (int i=0;i<N;i++){
64              out << randomwalk(Rrad) << endl;
65          }
66
67          // end timer
68          if (R[j] < 1.0){
69              end = clock();
70              out << "running time = " << (double)(end - begin)/(double)
                    CLOCKS_PER_SEC << " seconds." << endl;
71          }
72          else {
73              finish = time(0);
74              out << "running time = " << (double)(finish - start) << "
                    seconds." << endl;
75          }
76          out.close();
77      }
78      tend = time(0);
79      tim = tend - tstart;
80      cout << "running time = " << tim/60. << " minutes." << endl;
81      return 0;
82  }
```

Model kerapatan konstan – dengan MPI

```
1  #include <iostream>
2  #include <stdio.h>
3  #include <math.h>
4  #include <fstream>
5  #include <stdlib.h>
6  #include <time.h>
7  #include <ctime>
8  #include <string>
9  #include <mpi.h>
10 #define _USE_MATH_DEFINES
11 #define c 299792458.
12 using namespace std;
13
14 double unirand(){return (double) rand()/(double) RAND_MAX;}
15
16 double randomwalk(double Rrad){
17     double r, theta, phi, x, y, z, dx, dy, dz, f, rho, dt, t, rcol, randcol
           ;
18     double me = 9.11e-31; // kg
19     double mp = 1.67e-27; // kg
20     double sigmaT = 6.65245854533e-29; // m^2
21     unsigned int n=0;
```

```
22      x = 0.0; y = 0.0; z = 0.0, t=0.0, dt=0.0;
23      rho = 15000.; // Constant Density
24      f = (me+mp)/(sqrt(2.)*sigmaT*rho);
25      do{
26              phi = 2.*M_PI*unirand(); theta = acos(1.-2.*unirand());
27              do {
28                  randcol = unirand();
29              } while (randcol == 0.0 || randcol == 1.0); // mengindari -inf
30              rcol = -f*log(randcol);
31              dt = rcol/c;
32              t = t+dt;
33              dx = rcol*sin(theta)*cos(phi);
34              dy = rcol*sin(theta)*sin(phi);
35              dz = rcol*cos(theta);
36              x=x+dx;
37              y=y+dy;
38              z=z+dz;
39              n = n+1;
40              r = sqrt(x*x + y*y + z*z);
41      } while (r <= Rrad);
42
43      double tm = t/(Rrad/c);
44      return tm;
45  }
46
47  int main(int argc, char** argv){
48      time_t tstart=time(0), tend, tim;
49      int m = 8;
50      int N = 125;
51      double R[7]={0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0};
52
53          int rank, nprocs;
54          MPI_Init(&argc, &argv);
55          MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
56          MPI_Comm_rank(MPI_COMM_WORLD, &rank);
57
58      for (int k=0;k<7;k++){
59          double Rrad=R[k];
60          for (int j=0;j<m;j++){
61              if (rank == j){
62                  clock_t begin, end;
63                  time_t start, finish;
64                  if (Rrad < 1.0){ begin = clock();}
65                  else { start = time(0);}
66                  char filename[64];
67                  srand(time(NULL)+7+j);
68            sprintf(filename, "outfile-MPI-%d-%d.txt", k, j);
69                  ofstream out(filename);
70                  for (int i=0;i<N;i++){
71                      out << randomwalk(Rrad) << endl;
72                  }
73                  if (R[k] < 1.0){
74                      end = clock();
75                      out << "running time = " << (double)(end - begin)/(
```

```
                        double)CLOCKS_PER_SEC << " seconds." << endl;
76                  }
77                  else {
78                      finish = time(0);
79                      out << "running time = " << (double)(finish - start) <<
                            " seconds." << endl;
80                  }
81                  out.close();
82              }
83          }
84      }
85          MPI_Finalize();
86      tend = time(0);
87      tim = tend - tstart;
88      cout << "running time = " << tim/60. << " minutes." << endl;
89      return 0;
90  }
```

Model kerapatan linear – dengan MPI

```
1  #include <iostream>
2  #include <stdio.h>
3  #include <math.h>
4  #include <fstream>
5  #include <stdlib.h>
6  #include <time.h>
7  #include <ctime>
8  #include <string>
9  #include <mpi.h>
10 #define _USE_MATH_DEFINES
11 #define c 299792458.
12 using namespace std;
13
14 double unirand(){return (double) rand()/(double) RAND_MAX;}
15
16 double randomwalk(double Rrad){
17     double r, theta, phi, x, y, z, dx, dy, dz, f, rho, dt, t, rcol, randcol
           ;
18     double me = 9.11e-31; // kg
19     double mp = 1.67e-27; // kg
20     double sigmaT = 6.65245854533e-29; // m^2
21     unsigned int n=0;
22     x = 0.0; y = 0.0; z = 0.0, t=0.0, dt=0.0, r=0.0;
23     do{
24             rho = -3e4*5.*r/Rrad + 1.5e5; // linier gradient density
25             f = (me+mp)/(sqrt(2.)*sigmaT*rho);
26             phi = 2.*M_PI*unirand(); theta = acos(1.-2.*unirand());
27             do {
28                 randcol = unirand();
```

```
29              } while (randcol == 0.0 || randcol == 1.0); // menghindari -inf
30              rcol = -f*log(randcol);
31              dt = rcol/c;
32              t = t+dt;
33              dx = rcol*sin(theta)*cos(phi);
34              dy = rcol*sin(theta)*sin(phi);
35              dz = rcol*cos(theta);
36              x=x+dx;
37              y=y+dy;
38              z=z+dz;
39              n = n+1;
40              r = sqrt(x*x + y*y + z*z);
41      } while (r <= Rrad);
42
43      double tm = t/(Rrad/c);
44      return tm;
45  }
46
47  int main(int argc, char** argv){
48      time_t tstart=time(0), tend, tim;
49      int m = 8;
50      int N = 125;
51      double R[7]={0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0};
52
53          int rank, nprocs;
54          MPI_Init(&argc, &argv);
55          MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
56          MPI_Comm_rank(MPI_COMM_WORLD, &rank);
57
58      for (int k=0;k<7;k++){
59          double Rrad=R[k];
60          for (int j=0;j<m;j++){
61              if (rank == j){
62                  clock_t begin, end;
63                  time_t start, finish;
64                  if (Rrad < 1.0){ begin = clock();}
65                  else { start = time(0);}
66                  char filename[64];
67                  srand(time(NULL)+7+j);
68            sprintf(filename, "outfile-lin-MPI-%d-%d.txt", k, j);
69                  ofstream out(filename);
70                  for (int i=0;i<N;i++){
71                      out << randomwalk(Rrad) << endl;
72                  }
73                  if (R[k] < 1.0){
74                      end = clock();
75                      out << "running time = " << (double)(end - begin)/(
                            double)CLOCKS_PER_SEC << " seconds." << endl;
76                  }
77                  else {
78                      finish = time(0);
79                      out << "running time = " << (double)(finish - start) <<
                             " seconds." << endl;
80                  }
```

```
81                      out.close();
82                  }
83              }
84      }
85              MPI_Finalize();
86      tend = time(0);
87      tim = tend - tstart;
88      cout << "running time = " << tim/60. << " minutes." << endl;
89      return 0;
90  }
```